aiohttp-apispec

Dec 05, 2020

Contents:

Build and document REST APIs with aiohttp and apispec

`aiohttp-apispec` key features:

- `docs`, `request_schema`, `match_info_schema`, `querystring_schema`, `form_schema`, `json_schema`, `headers_schema`, `cookies_schema`, decorators to add swagger spec support out of the box;
- `validation_middleware` middleware to enable validating with marshmallow schemas from those decorators;
- **SwaggerUI** support.

`aiohttp-apispec` api is fully inspired by `flask-apispec` library

CHAPTER 1

Guide

1.1 Usage

1.1.1 Quickstart

Note: Using strict=True need only for marshmallow < 3.0.0

```
from aiohttp_apispec import (docs,
                             request_schema,
                             response_schema,
                             setup_aiohttp_apispec)
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')
    bool_field = fields.Bool()


class ResponseSchema(Schema):
    msg = fields.Str()
    data = fields.Dict()


@docs(tags=['mytag'],
      summary='Test method summary',
      description='Test method description')
@request_schema(RequestSchema(strict=True))
@response_schema(ResponseSchema(), 200)
```

(continues on next page)

(continued from previous page)

```
async def index(request):
    return web.json_response({'msg': 'done',
                              'data': {}})

# Class based views are also supported:
class TheView(web.View):
    @docs(
        tags=['mytag'],
        summary='View method summary',
        description='View method description',
    )
    @request_schema(RequestSchema(strict=True))
    def delete(self):
        return web.json_response({
            'msg': 'done',
            'data': {'name': self.request['data']['name']},
        })


app = web.Application()
app.router.add_post('/v1/test', index)
app.router.add_view('/v1/view', TheView)

# init docs with all parameters, usual for ApiSpec
setup_aiohttp_apispec(app=app, title="My Documentation", version="v1")


# find it on 'http://localhost:8080/api/docs/api-docs'
web.run_app(app)
```

1.1.2 Adding validation middleware

```
from aiohttp_apispec import validation_middleware

...

app.middlewares.append(validation_middleware)
```

Now you can access all validated data in route from `request['data']` like so:

```
@docs(tags=['mytag'],
      summary='Test method summary',
      description='Test method description')
@request_schema(RequestSchema(strict=True))
@response_schema(ResponseSchema(), 200)
async def index(request):
    uid = request['data']['id']
    name = request['data']['name']
    return web.json_response(
        {'msg': 'done',
         'data': {'info': f'name - {name}, id - {uid}'}}
     )
```

You can change `Request`'s `'data'` param to another with `request_data_name` argument of `setup_aiohttp_apispec` function:

```
setup_aiohttp_apispec(app=app,
                      request_data_name='validated_data',
                      title='My Documentation',
                      version='v1',
                      url='/api/docs/api-docs')

...

@request_schema(RequestSchema(strict=True))
async def index(request):
    uid = request['validated_data']['id']
    ...
```

1.1.3 More decorators

Starting from version 2.0 you can use shortenings for documenting and validating specific request parts like cookies, headers etc using those decorators:

Decorator name	Default put_into param
match_info_schema	match_info
querystring_schema	querystring
form_schema	form
json_schema	json
headers_schema	headers
cookies_schema	cookies

And example:

```
@docs(
    tags=["users"],
    summary="Create new user",
    description="Add new user to our toy database",
    responses={
        200: {"description": "Ok. User created", "schema": OkResponse},
        401: {"description": "Unauthorized"},
        422: {"description": "Validation error"},
        500: {"description": "Server error"},
    },
)
@headers_schema(AuthHeaders)
@json_schema(UserMeta)
@querystring_schema(UserParams)
async def create_user(request: web.Request):
    headers = request["headers"]  # <- validated headers!
    json_data = request["json"]  # <- validated json!
    query_params = request["querystring"]  # <- validated querystring!
    ...
```

1.1.4 Custom error handling

If you want to catch validation errors by yourself you could use `error_callback` parameter and create your custom error handler. Note that it can be one of coroutine or callable and it should have interface exactly like in examples below:

```
from marshmallow import ValidationError, Schema
from aiohttp import web
from typing import Optional, Mapping, NoReturn


def my_error_handler(
    error: ValidationError,
    req: web.Request,
    schema: Schema,
    error_status_code: Optional[int] = None,
    error_headers: Optional[Mapping[str, str]] = None,
) -> NoReturn:
    raise web.HTTPBadRequest(
            body=json.dumps(error.messages),
            headers=error_headers,
            content_type="application/json",
        )

setup_aiohttp_apispec(app, error_callback=my_error_handler)
```

Also you can create your own exceptions and create regular Request in middleware like so:

```
class MyException(Exception):
    def __init__(self, message):
        self.message = message

# It can be coroutine as well:
async def my_error_handler(
    error: ValidationError,
    req: web.Request,
    schema: Schema,
    error_status_code: Optional[int] = None,
    error_headers: Optional[Mapping[str, str]] = None,
) -> NoReturn:
    await req.app["db"].do_smth()  # So you can use some async stuff
    raise MyException({"errors": error.messages, "text": "Oops"})

# This middleware will handle your own exceptions:
@web.middleware
async def intercept_error(request, handler):
    try:
        return await handler(request)
    except MyException as e:
        return web.json_response(e.message, status=400)


setup_aiohttp_apispec(app, error_callback=my_error_handler)

# Do not forget to add your own middleware before validation_middleware
app.middlewares.extend([intercept_error, validation_middleware])
```

1.1.5 Named routes

Routes for the Swagger UI and to the swagger specification file *swagger.json* are registered as **'named resources <https://docs.aiohttp.org/en/stable/web_quickstart.html#reverse-url-constructing-using-named-resources'_** with the *swagger.docs* and *swagger.spec* names respectively. The corresponding routes are therefore avaialble un the

value returned by the application's router *named_resources()* call.

1.1.6 Build swagger web client

3.X SwaggerUI version

Just add `swagger_path` parameter to `setup_aiohttp_apispec` function.

For example:

Then go to `/docs` and see awesome SwaggerUI

2.X SwaggerUI version

`aiohttp-apispec` adds `swagger_dict` parameter to aiohttp web application after initialization (with `setup_aiohttp_apispec` function). So you can use it easily with `aiohttp_swagger` library:

```
from aiohttp_apispec import setup_aiohttp_apispec
from aiohttp_swagger import setup_swagger


def create_app(app):
    setup_aiohttp_apispec(app)

    async def swagger(app):
        setup_swagger(
            app=app, swagger_url='/api/doc', swagger_info=app['swagger_dict']
        )
    app.on_startup.append(swagger)
    # now we can access swagger client on '/api/doc' url
    ...
    return app
```

Now we can access swagger client on `/api/doc` url

1.2 Installation

```
$ pip install -U aiohttp-apispec
```

1.3 API Reference

aiohttp_apispec.**setup_aiohttp_apispec**(*app: aiohttp.web_app.Application*, *, *title: str = 'API documentation'*, *version: str = '0.0.1'*, *url: str = '/api/docs/swagger.json'*, *request_data_name: str = 'data'*, *swagger_path: str = None*, *static_path: str = '/static/swagger'*, *error_callback=None*, *in_place: bool = False*, *prefix: str = ''*, *schema_name_resolver: Callable = <function resolver>*, ***kwargs*) → None

aiohttp-apispec extension.

Usage:

```
from aiohttp_apispec import docs, request_schema, setup_aiohttp_apispec
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')
    bool_field = fields.Bool()


@docs(tags=['mytag'],
      summary='Test method summary',
      description='Test method description')
@request_schema(RequestSchema)
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})


app = web.Application()
app.router.add_post('/v1/test', index)

# init docs with all parameters, usual for ApiSpec
setup_aiohttp_apispec(app=app,
                      title='My Documentation',
                      version='v1',
                      url='/api/docs/api-docs')

# now we can find it on 'http://localhost:8080/api/docs/api-docs'
web.run_app(app)
```

Parameters

- **app** (*Application*) – aiohttp web app
- **title** (*str*) – API title
- **version** (*str*) – API version
- **url** (*str*) – url for swagger spec in JSON format
- **request_data_name** (*str*) – name of the key in Request object where validated data will be placed by validation_middleware (`'data'` by default)
- **swagger_path** (*str*) – experimental SwaggerUI support (starting from v1.1.0). By default it is None (disabled)
- **static_path** (*str*) – path for static files used by SwaggerUI (if it is enabled with `swagger_path`)
- **error_callback** – custom error handler
- **in_place** – register all routes at the moment of calling this function instead of the moment of the on_startup signal. If True, be sure all routes are added to router
- **prefix** – prefix to add to all registered routes
- **schema_name_resolver** – custom schema_name_resolver for MarshmallowPlugin.
- **kwargs** – any apispec.APISpec kwargs

aiohttp_apispec.**docs**(***kwargs*)

Annotate the decorated view function with the specified Swagger attributes.

Usage:

```
from aiohttp import web

@docs(tags=['my_tag'],
      summary='Test method summary',
      description='Test method description',
      parameters=[{
              'in': 'header',
              'name': 'X-Request-ID',
              'schema': {'type': 'string', 'format': 'uuid'},
              'required': 'true'
          }]
      )
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})
```

aiohttp_apispec.**request_schema**(*schema*, *locations=None*, *put_into=None*, *example=None*, *add_to_refs=False*, ***kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**match_info_schema**(*schema*, ***, *locations=['match_info']*, *put_into='match_info'*, *example=None*, *add_to_refs=False*, ***kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs

arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**querystring_schema**(*schema, *, locations=['querystring'], put_into='querystring', example=None, add_to_refs=False, **kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance

- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**form_schema**(*schema, *, locations=['form'], put_into='form', example=None, add_to_refs=False, **kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**json_schema**(*schema, *, locations=['json'], put_into='json', example=None, add_to_refs=False, **kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')
```

(continues on next page)

(continued from previous page)

```
@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**headers_schema**(*schema, *, locations=['headers'], put_into='headers', example=None, add_to_refs=False, **kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**cookies_schema**(*schema, *, locations=['cookies'], put_into='cookies', example=None, add_to_refs=False, **kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**response_schema**(*schema*, *code=200*, *required=False*, *description=None*)

Add response info into the swagger spec

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class ResponseSchema(Schema):
    msg = fields.Str()
    data = fields.Dict()

@response_schema(ResponseSchema(), 200)
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})
```

Parameters

- **description** (*str*) – response description
- **required** (*bool*) –
- **schema** – Schema class or instance

- **code** (*int*) – HTTP response code

aiohttp_apispec.**use_kwargs**(*schema*, *locations=None*, *put_into=None*, *example=None*, *add_to_refs=False*, ***kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse
- **put_into** – name of the key in Request object where validated data will be placed. If None (by default) default key will be used
- **example** (*dict*) – Adding example for current schema
- **add_to_refs** (*bool*) – Working only if example not None, if True, add example for ref schema. Otherwise add example to endpoint. Default False

aiohttp_apispec.**marshal_with**(*schema*, *code=200*, *required=False*, *description=None*)

Add response info into the swagger spec

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class ResponseSchema(Schema):
    msg = fields.Str()
    data = fields.Dict()

@response_schema(ResponseSchema(), 200)
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})
```

Parameters

- **description** (*str*) – response description
- **required** (*bool*) –

- **schema** – Schema class or instance
- **code** (*int*) – HTTP response code

aiohttp_apispec.**validation_middleware**(*request: aiohttp.web_request.Request*, *handler*) → aiohttp.web_response.Response

Validation middleware for aiohttp web app

Usage:

```
app.middlewares.append(validation_middleware)
```

Python Module Index

a

Index